

СОДЕРЖАНИЕ

1. ЦЕЛЬ РАБОТЫ.....	4
2. КОМПОНЕНТЫ ВЫБОРА И НАСТРОЙКИ ПАРАМЕТРОВ.....	4
3. ИСПОЛЬЗОВАНИЕ ПОЛОС ПРОКРУТКИ ДЛЯ ВВОДА ИНФОРМАЦИИ	6
4. ИСПОЛЬЗОВАНИЕ СПИСКОВ.....	7
5. РАБОТА С ОКНАМИ ДИАЛОГА.....	9
6. КОМПОНЕНТЫ УПРАВЛЕНИЯ ФАЙЛАМИ	11
7. ПРАКТИЧЕСКОЕ ЗАДАНИЕ	12
ПРИЛОЖЕНИЕ А (справочное).....	14

1. ЦЕЛЬ РАБОТЫ

Целью работы является практическое освоение методологии и принципов создания элементов выбора и диалога как стандартных компонент интерфейса Windows-программы в среде визуального проектирования.

2. КОМПОНЕНТЫ ВЫБОРА И НАСТРОЙКИ ПАРАМЕТРОВ

Выбор и настройка параметров при работе с программным приложением считается стандартной частью работы пользователя с любым приложением. Это может быть как настройка самого приложения, так и определение параметров отображаемых или моделируемых в приложении процессов и явлений. Элементы интерфейса Windows-программы для основных операций такой работы в настоящее время практически стандартизированы. Рассмотрим создание этих элементов на примере работы с компонентами библиотеки VCL (Visual Component Library) в среде Delphi.

Базовые элементы выбора и настройки параметров расположены на странице Standard палитры компонент Delphi. В представленном ниже проекте используем следующий классический набор компонент:



GroupBox – группа, которая визуально и логически объединяет наборы компонент, определяют порядок перемещения по компонентам на форме (при нажатии клавиши TAB). При помещении в группу новый компонент получает свойства `ParentColor`, `ParentShowHint`, `ParentFont`, `ParentCtl3D` этой группы. Свойства `Left` и `Top` сгруппированных объектов определяются по верхнему углу группы, а не формы;



RadioGroup – группа для объектов **RadioButton** (см. ниже);



RadioButton – переключатели или радиокнопки, служат для выбора одной возможности из набора взаимоисключающих возможностей. Термин отражает сходство с набором кнопок выбора каналов радиоприемника. Эти кнопки обычно объединяют группой **RadioGroup**. Выбор кнопки отражает свойство `Checked`, свойство `Alignment` определяет положение поясняющей надписи относительно кнопки;



CheckBox – выключатель, выглядит как строка текста с окошком для установки отметки о выборе. Выключатели работают независимо, но их обычно группируют. При определении

реакции на выбор можно использовать событие OnClick, но обычно устанавливают как индикатор свойство State по трем состояниям: cbChecked (есть), cbUnChecked (нет), cbGrayed (неопределенно) внутри программы. При этом для блокировки ручного изменения этого свойства нужно установить DragMode = dmAutomatic.

Пример проекта с выбором параметров.

Установим на форме (Рисунок 2.1) компонент RadioGroup из четырех радиокнопок – «красный», «желтый», «зеленый», «готово». Установим кнопку Exit со свойством Enabled = False. В реакцию OnClick этой кнопки можно ввести, например, Close, в радиокнопки установки цвета формы – строки типа:

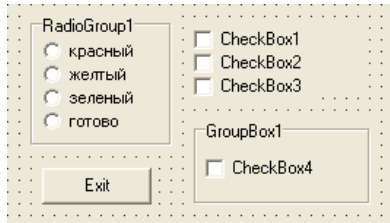


Рисунок 2.1

```
Button1.Enabled := False;
```

```
Form1.Color := clRed;
```

А для кнопки «готово»:

```
Button1.Enabled := True;
```

При этом включение радиокнопок будет либо менять цвет формы, либо активизировать кнопку «Exit».

Добавим на форму три переключателя и три метки Label (справа сверху на Рисунок 2.1) и введем смену цвета меток при переключениях:

```
if Label1.Color <> clRed
then Label1.Color := clRed
else Label1.Color := Form1.Color;
Test(Label1, Label2, Label3, CheckBox4, Sender);
```

Здесь вызывается процедура Test для анализа «включенных» меток в дополнительном переключателе (CheckBox4 с именем «два фонаря»).

```
Test(Label1, Label2, Label3: TLabel; CheckBox4: TCheckBox;
Sender: TObject);
{ подсчет числа «фонарей» с отображением в CheckBox5 }
var r, r1, r2, r3: integer;
begin
  r1 := 0; r2 := 0; r3 := 0;
  if Label1.Color = clRed then r1 := 1;
```

```

if Label2.Color = clYellow then r2 := 1;
if Label3.Color = clGreen then r3 := 1;
r := r1+r2+r3;
if r < 2 then CheckBox4.State := cbUnchecked { отключено }
else if r = 2 then CheckBox4.State := cbChecked { включено }
else CheckBox4.State := cbGrayed { нейтраль }
end;

```

В зависимости от состояния меток переключатель CheckBox4, созданный со свойством DragMode = dmAutomatic, будет находиться в одном из трех состояний по свойству State.

3. ИСПОЛЬЗОВАНИЕ ПОЛОС ПРОКРУТКИ ДЛЯ ВВОДА ИНФОРМАЦИИ

Полоса прокрутки – классический элемент оконного интерфейса, использующийся для скроллинга информации, которая не помещается целиком в область вывода. В объектах просмотра и редактирования Windows этот элемент появляется при необходимости автоматически. В палитре VCL-компонент для создания полос прокрутки как самостоятельного элемента введен отдельный компонент:



ScrollBar – Самостоятельная полоса прокрутки, может быть горизонтальной или вертикальной, что определяется свойством Kind = sbHorizontal/sbVertical. Свойствами Min и Max можно задать края диапазона целых значений, соответствующих положению бегунка полосы (свойство Position). Обычно также выбирают размеры скачков бегунка при щелчке на стрелках или на свободной полосе, определяя свойства SmallChange, LargeChange.

При перемещении бегунка ScrollBar генерируется сообщение OnChange (при попытке перемещения – OnScroll). При этом можно получать и обрабатывать данные как свойство Position. Во время работы программы значения свойств Position, Min, Max можно задавать с помощью метода SetParams.

Рассмотрим пример демонстрации работы RGB-функций (установки цвета по трем составляющим) с помощью полос прокрутки.

На форме (Рисунок 3.1) разместим три полосы прокрутки, дав им имена RedBar, GreenBar, BlueBar и установив

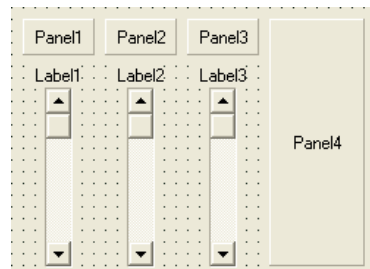


Рисунок 3.1

свойство `Max = 255` по количеству градаций компонент RGB. Поместим три соответствующие полосам метки (`Label1-3`) и панели (`Panel1-3`). Каждый бегунок должен будет менять вклад RGB-компонента, отображающийся на панели как цвет, а на метке – как число. Результирующий цвет отобразим на панели `Panel4`.

Зададим реакцию `OnChange` бегунков, например для синего:

```
Panel3.Color := TColorRef(RGB(0,0,BlueBar.Position));
Label3.Caption := IntToStr(BlueBar.Position);
Panel4.Color := TColorRef(RGB(
    RedBar.Position,
    GreenBar.Position,
    BlueBar.Position));
```

Функция `TColorRef` преобразует значения цветовых составляющих в число (типа `LongInt`), а функция `IntToStr` преобразует вес компонента (определяемый положением бегунка) в строку, отображаемую на метках.

Вызов всех этих функций желательно также поместить в событие при создании формы (`OnCreate` для `Form1`), а начальное положение бегунков определить при этом как среднее (`Position = 122`).

4. ИСПОЛЬЗОВАНИЕ СПИСКОВ



`ListBox` – обычный список, этот компонент предназначен для работы с перечнем текстовых элементов (с ограничением по количеству до ~5000 шт). Перечень можно создавать (в том числе загружать как строки из текстового файла), преобразовывать и выгружать в файл. Элементы списка могут быть выбраны с помощью клавиатуры или мыши. Классический пример использования `ListBox` в среде Windows – выбор файла из списка в пункте меню `File/Open` многих приложений.

Основное свойство списка – `Items` (массив строк), оно аналогично свойству `Lines` для компонента `Memo`. Индекс выбранного элемента списка хранится в переменной `ItemIndex`. Методы `Add`, `Delete`, `Insert` используются для добавления, удаления и вставки строк.

Свойство `Sorted = True` упорядочивает список по возрастанию кода символов строк. `ItemHeight` – вертикальный размер элементов, `Columns` – число колонок в списке, `ExtendedSelect` – возможность множественного выбора элементов (при удержании клавиши `Shift`), при этом для выбранных элементов свойство `Selected = True`.



ComboBox – комбинированный список, дополнительно к обычному включает строку ввода. Из нескольких типов ComboBox наиболее популярен спадающий вниз (drop-down combo box).

Создадим типовый проект с компонентом ListBox.

На форме (Рисунок 4.1) кроме списка разместим ряд кнопок (или пунктов меню), а также две строки ввода Edit1, Edit2 и две метки Label1, Label2. По выбору пунктов организуем следующие операции со списком:

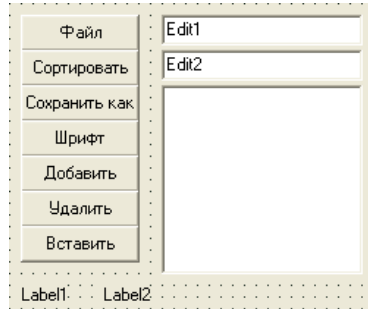


Рисунок 4.1

Загрузка строк из файла, имя которого предварительно набирается в строке ввода (пункт «Файл»):

```
ListBox1.Sorted := false;  
ListBox1.Items.LoadFromFile(Edit1.Text);
```

Сортировка списка (пункт «Сортировать»):

```
ListBox1.Sorted := true;
```

Запись списка в файл, имя которого предварительно набирается в строке ввода (пункт «Сохранить как»):

```
ListBox1.Items.SaveToFile(Edit2.Text);  
MessageDlg('Создан файл '+Edit2.Text, mtInformation, [mbOK], 0);
```

Загрузка списка экранных шрифтов (пункт «Шрифт»):

```
ListBox1.Items := Screen.Fonts;
```

Добавление случайного числа в список с соблюдением сортировки, если она задана (пункт «Добавить»):

```
var  
  s: string;  
begin  
  str(random:10:8,s); { генерация случайного числа }  
  ListBox1.Items.Add(s);  
end;
```

Добавление числа в нужное место списка (пункт «Вставить»):

```

var s: string;
begin
  str(random:10:8,s); { генерация случайного числа }
  ListBox1.Items.Insert(ListBox1.ItemIndex,s);
end;

```

Удаление выбранного элемента списка (пункт «Удалить»):

```
Listbox1.Items.Delete(Listbox1.ItemIndex);
```

Выведем некоторые характеристики выбранного элемента на метках:

```

var
  code: integer;
  a: real;
begin
  Label2.Caption := ListBox1.Items[ListBox1.ItemIndex];
  val(Label2.Caption,a,code);
  if code = 0 then Label1.Caption := 'число'
    else Label1.Caption := 'строка';
end;

```

5. РАБОТА С ОКНАМИ ДИАЛОГА

Палитра компонент Delphi содержит закладку Dialogs – диалоги работы с текстовыми и графическими файлами (открытие и сохранение), выбор цвета и шрифта, поиск и замена, работа с принтером (Рисунок 5.1).

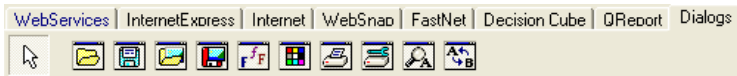


Рисунок 5.1

Объекты, представленные здесь, невидимы во время выполнения программы, – окна диалога активизируются лишь при определенных событиях, задаваемых в проекте. Чаще всего это выбор команды меню или нажатие кнопки. Характеристики и свойства диалоговых компонент приведены в приложении А.

Обычно окна диалога используются в солидных проектах с переработкой информации из файлов различных типов. Поэтому в качестве учебного примера создадим

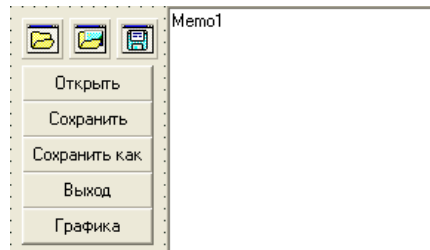


Рисунок 5.2

проект из двух форм – основной (Form1, свойство FormStyle = fsMDIForm) и дочерней (Form2, FormStyle = fsMDIChild). Дочернюю форму введем из меню File/New и затем добавим ее в проект.

На основной форме (Рисунок 5.2) разместим кнопки «Открыть», «Сохранить», «Сохранить как», «Выход» и «Графика», а также поле Memo с Align = alRight. Здесь же поместим три диалога – два OpenFileDialog (один для текста, второй – для графики) и один SaveDialog.

На дочерней форме разместим компонент Image (с закладки Additional) для вывода рисунков. Свойство Align = alClient определит заполнение по краям формы, а свойство Stretch – растяжку рисунка по границам.

Для кнопки «Открыть» введем загрузку в поле примечаний содержимого файла:

```
with OpenFileDialog1 do
  if Execute then begin
    Memo1.Visible := true; { видимость поля редактора }
    Memo1.Lines.LoadFromFile(FileName);
    Caption := 'Мой редактор ' + ExtractFileName(FileName);
    SaveDialog1.FileName := FileName;
    FileName := '';
  end;
```

Для кнопки «Сохранить»:

```
Memo1.Lines.SaveToFile(SaveDialog1.FileName);
```

Для кнопки «Сохранить как»:

```
with SaveDialog1 do
  if Execute then begin
    Memo1.Lines.SaveToFile(FileName);
    Caption := 'Мой редактор ' + ExtractFileName(FileName);
  end;
```

Для кнопки «Графика» зададим деактивацию поля Memo для освобождения пространства главной формы:

```
with OpenFileDialog2 do
  if Execute then begin
    Memo1.Visible := false;
    Screen.Cursor := crHourGlass; { курсор "песочные часы" }
    with Form2.Image1.Picture do
```



```

LoadFromFile(FileName);
Caption := ExtractFileName(FileName);
Screen.Cursor := crDefault; { нормальный курсор }
end;

```

Принцип использования любого стандартного окна диалога одинаков – вызывается его метод `Execute` и присваиваются возвращаемые им значения свойствам тех компонент, на которые они влияют.

Для нормальной работы диалоговых компонент необходимо определять свойство `Filter` (двойным щелчком в инспекторе объектов), например для диалогов с текстовыми файлами обычно заполняют две строки «Текстовые файлы – *.txt» и «Все файлы – *.*». Для графических файлов можно определить «Растры – *.bmp», «Пиктограммы – *.ico», «Метафайлы – *.wmf».

6. КОМПОНЕНТЫ УПРАВЛЕНИЯ ФАЙЛАМИ

Описанные выше диалоговые панели работы с файлами общего назначения (тип `OpenDialog` и `SaveDialog`) часто несут избыточную информацию, например, когда требуется только список файлов текущего каталога или список логических устройств. В этом случае используют простые файловые компоненты с закладки Win 3.1. Таких компонент здесь четыре:



`FileListBox` – список файлов указанного каталога;



`DirectoryListBox` – список каталогов указанного диска;



`DriveComboBox` – список логических устройств;



`FilterComboBox` – задание шаблона для `FileListBox`.

Приведем пример проекта работы с файлами с использованием файловых компонент.

Поместим перечисленные выше

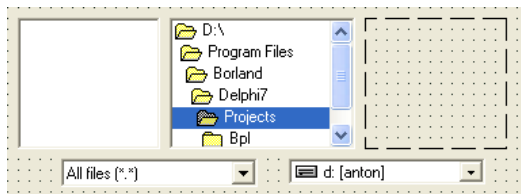


Рисунок 6.1

компоненты на форму вместе с компонентом Image (Рисунок 6.1) для просмотра выбранных из списка файлов, содержащих графические изображения.

Связываем компоненты между собой в инспекторе объектов, для этого установим следующие свойства компонентов:

```
DirectoryListBox1.FileList = FileListBox1;  
FilterComboBox1.FileList = FileListBox1;  
DriveComboBox1.DirList = DirectoryListBox1;
```

Свойство Filter шаблона файлов (FilterComboBox) определим при создании формы:

```
FilterComboBox1.Filter := 'Метафайлы|*.wmf|'  
    + 'Иконки|*.ico|'  
    + 'Растры|*.bmp';
```

Выбранный файл должен отображаться в области компонента Image1, поэтому в FileListBox на событие OnChange вводим

```
if FileListBox1.FileName <> '' then  
    Image1.Picture.LoadFromFile(FileListBox1.FileName);
```

7. ПРАКТИЧЕСКОЕ ЗАДАНИЕ

7.1. Составить проект для визуализации выбираемого стиля, размера и цвета шрифта. Сам шрифт как набор всех латинских и русских букв (как прописных, так и строчных) отображать на метке. Каждую характеристику шрифта выбирать из набора минимум четырех радиокнопок.

7.2. Составить проект для анализа введенной в строке Edit информации: текстовая, числовая, прочая. В качестве индикаторов использовать набор из трех компонент CheckBox. Ввести четвертый индикатор для анализа очередного набираемого символа.

Предусмотреть кнопки «Новые данные» и «Выход». Всем введенным компонентам задать ярлычки с оперативной подсказкой (Hints). При оформлении компонент использовать по возможности различные цвета и шрифты.

7.3. Составить проект для нахождения целочисленных решений уравнения $X^2 + Y^2 = R^2$, то есть точек, с целочисленными координатами, лежащих на окружности радиуса R. Использовать три компонента ScrollBar, первый из которых будет определять радиус в диапазоне от 5

до 25, а два других – варьировать величины X и Y от 0 до R . Величины X , Y , R , а также погрешность в решении уравнения выводить на метках. Ввести индикатор нахождения решения.

7.4. Составить проект для работы со списком, аналогичный описанному в разделе 4, но с использованием компонента `ComboBox`. При этом создать текстовый файл, содержащий минимум 20 строк, например, фамилии студентов. Отображать длину выбранного элемента списка.

7.5. Составить проект «Редактор текстового файла» с использованием компонента `ListBox`. Имя загружаемого и сохраняемого файла берется из строк ввода (`Edit`). Предусмотреть кнопки «Очистка строк ввода», «Сохранить», «Сохранить как» и «Выход».

7.6. Модернизировать п. 7.5, введя второй компонент `ListBox` для имитации двухоконного редактора файлов. Ввести также окна сообщений для подтверждения проводимых в проекте операций.

7.7. Составить проект с использованием окон диалога `OpenDialog`, `SaveDialog`, `FontDialog`, `ColorDialog`, `FindDialog` и `ReplaceDialog` для работы с текстовым файлом, отображаемым в поле `Memo`.

7.8. Составить проект для работы с файлами, аналогичный описанному в разделе 6, но для текстовых файлов с расширениями «`pas`», «`txt`», «`bak`».

ПРИЛОЖЕНИЕ А

(справочное)

Таблица А.1 Компоненты, реализующие стандартные панели диалога











	OpenDialog	Выбор открываемого файла по шаблону
	SaveDialog	Создание файла
	FontDialog	Выбор шрифта и его характеристик
	ColorDialog	Выбор цвета
	PrintDialog	Вывод на устройство печати
	PrinterSetupDialog	Панель настройки устройства печати
	FindDialog	Панель поиска
	ReplaceDialog	Панель замены
	OpenPictureDialog	Выбор графического изображения с просмотром
	SavePictureDialog	Сохранение графического изображения с просмотром

Таблица А.2 Общие свойства диалоговых компонент

DefaultExt	Расширение имени файла, которое подставляется в файл, если пользователь указал только имя файла.
FileEditStyle	Тип строки редактирования для ввода имени файла: редактор (fsEdit) или список (fsComboBox – позволяет указать тип файла по свойству List).
Filter	Шаблон имен файлов, отображаемых в панели диалога. Например, для текстовых файлов: Filter = '*.txt' или с описанием: Filter = 'Текстовые файлы *.txt'.
FilterIndex	Номер активного шаблона из заданных для списка шаблонов (по умолчанию равен 1).
InitialDir	Имя каталога, содержимое которого отображается при вызове (по умолчанию используется текущий каталог).
Options	Опции, отражающие специфику работы с файловой системой или выбирающие стиль оформления объектов.
Title	Текст заголовка диалоговой панели.